

# 1. Série domácích úkolů

Odevzdávejte do 27. 10. 14:00

1. *Naivní algoritmus může být pomalý:* [5b]  
 Naivní algoritmus, který zkouší všechny možné začátky jehly v seně a vždy porovnává řetězce, má časovou složitost  $\mathcal{O}(JS)$ . Může být opravdu tak pomalý, uvážíme-li, že porovnávání řetězců skončí, jakmile najde první neshodu? Sestrojte vstup, na kterém algoritmus poběží  $\Theta(JS)$  kroků, přestože nic nenajde.
  
2. *Nejdelší vlastní prefix, který je sufixem:* [5b]  
 Je dáno slovo. Chceme nalézt jeho nejdelší vlastní prefix, který je současně sufixem.
  
3. *Příliš mnoho jehel:* [5b]  
 Nalezněte příklad jehel a sena, v němž je asymptoticky více než lineární počet výskytů. Přesněji řečeno ukažte, že pro každé  $n$  existuje vstup, v němž je součet délek jehel a sena  $\Theta(n)$  a počet výskytů není  $\mathcal{O}(n)$ .
  
4. *Počet výskytů:* [5b]  
 Mějme seno a jehly. Popište algoritmus, který v lineárním čase pro každou jehlu spočítá, kolikrát se v seně vyskytuje. Časová složitost by neměla záviset na počtu výskytů – ten, jak už víme, může být superlineární.

1) I když porovnání dvou řetězců ukončíme po první neshodě:

abc  
bbc  
 ↑

tak bude výsledná složitost následující:

| Délka prefixu jehly           | počet porovnání |
|-------------------------------|-----------------|
| 1                             | $S$             |
| 2                             | $\frac{S}{2}$   |
| 3                             | $\frac{S}{3}$   |
| ⋮                             | ⋮               |
| $\lfloor \frac{J}{2} \rfloor$ | $\frac{S}{J}$   |

*-> uvažujeme, že vždy dojde k neshodě hned v prvním písmenu*

*Což je sice hrubě ale přesně  $\mathcal{O}(S)$ .*

$$\sum_{i=1}^J \frac{S}{i} = S \cdot \sum_{i=1}^J \frac{1}{i} = S \cdot H_J \approx S \cdot \log J$$

*Za předpokladu, že  $H_J := J$ -tí harmonické číslo, které hrubě odhaduje  $\log$*

Vstup, pro který by minimální alg. vyšel časově  $\Theta(|S|)$ :

Řešení, které bude k-násobkem  $|S|$ , tedy  $S = k \cdot |S|$

Vpří:  $S = abcb \dots ab$

$S = \underbrace{abcb \dots ab}_{k \text{ krát}}$

$O(|S|)$

Prů:  $\left. \begin{array}{l} \text{Délka prefixu je} \\ 1 \\ 2 \\ 3 \\ \vdots \\ |S| \end{array} \right\}$

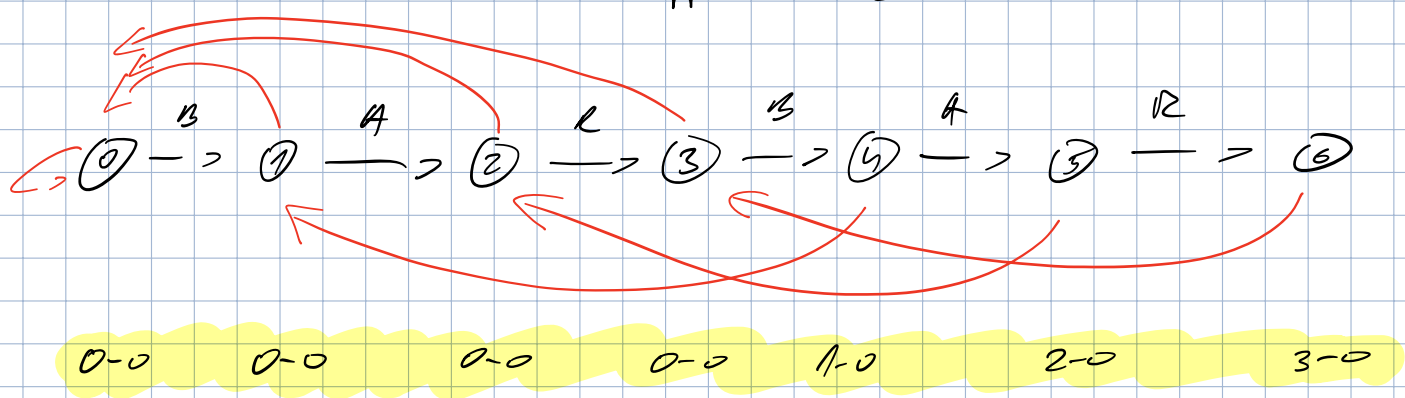
$\left. \begin{array}{l} \text{Délka prefixu je} \\ 1 \\ 2 \\ 3 \\ \vdots \\ |S| \end{array} \right\}$

# porovnání  
 1.  $|S|$   
 2.  $\frac{|S|}{2}$   
 3.  $\frac{|S|}{3}$   
 ...  
~~5.  $\frac{|S|}{5}$~~

2) Nejdelší vlastní prefix, co je suffix:

BARBAR → LPS = BAR

incl. ↓  
 excl. ↓  
 prefix :=  $S[i+1:]$   
 suffix :=  $S[:i]$



Zpětná hmna z posledního stavu UMP automatu povede na nejblíže další výskyt takového prefixu. Pokud bude sláhat po zpětných hmnách z poslední pozice UMP automatu a bude sčítat rozdíly indexů stavů, dostaneme index nejdelšího prefixu, co je zároveň suffix.

Stážitost algoritmu je  $O(|S|)$ , jelikož  $O(|S|)$  trvá vytvořit UMP automat a přičet po zpětných hmnách z posledního stavu musí být také nejvýše  $O(|S|)$ .

Níže příložený kód:

```
def Krok(i: int, x: str, input_string: str, z: array):  
    while input_string[i] != x:  
        if (i == 0):  
            return 0  
        i = z[i]  
    return i+1
```

```
def VytvorKMP(input_string: str):  
    z = [0]*(len(input_string)+1)  
    z[0] = 0  
    z[1] = 0  
    i = 0  
    for j in range(2, len(input_string)+1):  
        i = Krok(i, input_string[j-1], input_string, z)  
        z[j] = i  
    return z
```

```
def NajdiNejdelsiPrefixCojeSuffix(z: array, input_string: str):  
    t = z[len(z)-1]  
    final_len = 0  
    while (t != 0):  
        final_len += t - z[t]  
        t = z[t]  
    return input_string[:final_len]
```

→ Skokování zpět po zpětných hranách  
→ uhládkování se rozdělují indexu pro finální délku prefixu.

```
retezec = input("Zadejte retezec: \n")  
print(NajdiNejdelsiPrefixCojeSuffix(VytvorKMP(retezec),retezec))
```

3) Příliš mnoho jehol

$$S = \underbrace{\text{"AAA...AA"}}_n$$

$$J = \left\{ \underbrace{\text{"AA...AA"}}_{n/2}, \underbrace{\text{"A...A"}}_{n/4}, \underbrace{\text{"A...A"}}_{n/8}, \dots \right\}$$

→ dokud  $\frac{n}{2^i} > 1$

$$|S| = n$$
$$|J| = \sum_{i=1}^{\infty} \frac{n}{2^i} \leq n \cdot \sum_{i=1}^{\infty} \frac{1}{2^i} = n \cdot 1$$

$S, J \in O(n)$ , ale výsledky rostou rychleji:

Délka prefixu sama

# výsledků jehol

- 1
- 2
- 3
- 4
- 5
- 6

- 1
- 3
- 6
- 10
- 15
- 21

→ Avšak počet výsledků pro takové jehlo roste očividně rychleji jako  $O(n)$

4) Počet výskytů jehly  $\rightarrow$  lineární přístup

Pokud jde o lineární přístup vzhledem k seznamu a jednotlivé jehle,  
lze využít DMP alg., letený pracuje v čase  $O(N+S)$ .

Ten pracuje tak, že v čase  $O(N)$  vytvoří vyhledávací automat pro danou jehlu:

to dvěma přechody, kdy jednotlivé sestojící dopředné hranice a následně v  
druhém přechodu pustím na stejný automat substring jehly  $J[1:]$  a

bude si pamatovat stav, do kterých jsem vstoupil. Tím efektivně vytvořím epótní hranice a funkce i  
v čase  $O(N)$  celý vyhledávací automat pro jednotlivou jehlu.

Následně pro zadané seno státní automat projít a vždy, když dojde

do koncového stavu automatu, inkrementovat # výskytů, letený po celém přechodu seznamu

vratím jako finální počet výskytů. Celkem to pro každou jehlu bude práce  $O(N+S)$ .